Available online at www.sciencedirect.com

SciVerse ScienceDirect

International Meshing Roundtable 00 (2016) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

25th International Meshing Roundtable (IMR25)

All hexahedral boundary layers generation

Loïc Maréchal[a]

[a]*INRIA, 1 rue Honoré d'Estienne d'Orves, 91120 Palaiseau, France*

Abstract

Boundary layers generation has recently become one of the most sought-after features in mesh generation. But behind this single feature hides a lot of different capacities that range from the most basic, like extruding a layer from a surface, to imprinting, cross-imprinting and multi-normals capabilities that are much more challenging to develop. This paper aims at implementing all those features using hexahedral elements only and is based on a previously developed automated hexahedral mesh generator based on the octree method. Both points, octree and hexahedra, implying some constraints that this paper will address in a very pragmatic way. Indeed, we view automatic meshing from the industrial user's point of view: *no mesh, no run*, consequently this work's philosophy has been always to favor automaticity and genericity over element's quality and geometric approximation.

Keywords: octree, hexahedral mesh, boundary layers, imprinting, cross-imprinting

1. Boundary-layers: what it is all about

Generating boundary layers (BL hereafter) consists in growing layers of stretched elements from a surface mesh.

In the early years of mesh generation, and up to a time not so distant, that was all BL where about. But when this simple feature became widely available and used in numerical simulations, a truckload of requests came from the users like multi-normals, normal-blending, imprinting, cross-imprinting, blending layers, and so on.

This section aims at listing and shortly describing all the desired features that should be implemented in an industrial grade automated mesher.

Source control. We want to grow layers from some specific surface references or patches and may want to specify a different set of layer parameters on each of those references or groups of references.

Layers parameters. For each source reference we would like to specify the number of layers, their size and geometrical growth factor.

* Corresponding author. Tel.: +33-177-578-005
E-mail address: loic.marechal@inria.fr

Ending mode. Layers may not cover the whole surface and may stop at some surface patch limits. We would like to specify whether the layers should vanish with wedge shaped elements or should *imprints* the surface it stops at. This parameter could be set by the user or automatically derived from the dihedral angle between the two surface quads that belong to the different domains. In case of wide angle, a wedge ending is preferred and when the angle is sharper than 90 degrees, imprinting is preferred, in both cases for the sake of elements quality.

Cross-imprinting. Two sets of layers growing from neighbor surface patches, may fuse or intersect each other, creating a *cross-imprinting* pattern.

Normal blending. Layers should grow orthogonally to the surface, but in some concavities, it may create distorted or invalid elements. This problem may be circumvented by slightly bending the normal vectors, skewing the elements, to preserve their quality.

Blending layers. The last layer of a set is connected directly to the volume mesh and may be so thin that a strong volume shock could happen, leading to imprecise numerical results. A set of extra layers, whose thickness should grow from the final *physical* layer up to the background element size, may be added to mitigate the size transition. Hence the name *blending* layers.

Multi-normals. When layers have to be grown from a very sharp convex angle, they tend to generate very poorly shaped elements. A nice way to deal with that is to round-up the sharp angle to preserve the mesh quality.

In this paper we will try to address each point as best as we can given the strong constraints coming from the octree structure, which suffers from its inherent isotropy and axis alignment, and the hexahedral elements that are much more rigid, topologically and geometrically than tetrahedra.

We know beforehand that we will not be able to reach the tetrahedral BL capacities that are much higher, thanks to the nimble nature of simplicial elements over hexahedra. The greatest capacities that are difficult, if not impossible, to implement with octree and hexahedra, are anisotropic meshing as is seen in [12], the ability to push the layers very far away from the boundary as seen in [9] and far stretching cross-imprinting as done in [11].

However, some interesting work has been done on hexahedral boundary layers generation using sheet insertion [7], smoothing and extrusion [10] and even non conformal meshes based on octree [8].

The philosophy behind this work is always to favor the automatic and generic aspects of the algorithm since it is intended to be used as a *black box* software component. So whatever the input surface is, and whatever the BL parameters the user requests, the method must always produce a valid result, be it different from what the user is expecting in some complex cases.

In other words, the constant parts of the equation are the automaticity and genericity and the variables are the elements quality and the geometric accuracy.

2. Octree based hexahedral meshing: a quick overview

The present work is based on *Hexotic*, an automated hexahedral-mesh generator based on the octree method that was presented at the IMR 2009 conference. Since the objective of this work is to focus on BL generation only, the reader is encouraged to read this previous paper for a deeper overview. For the sake of understanding, the mesher's basic principles are shortly described in this section. The process can be broken-down into the following steps:

1. refine an octree structure according to the input surface meshes local feature sizes,
2. derive a dual mesh from the octree,
3. color volume subdomains separated by boundary faces,
4. insert a buffer layer of elements near the surface,
5. project boundary nodes on a smoothed surface while displacing the other inner nodes for the sake of quality,
6. insert a buffer layer of elements near the sharp angles,
7. project the boundary nodes on the real surface with sharp features while smoothing the other nodes.

Such algorithm suffers from many drawbacks: it is isotropic, thus generating too many elements, and its geometric approximation is rather poor, especially near sharp angles. However, it is completely automated and can be integrated as a black-box component in any numerical simulation software suite. It is also very fast (millions of elements per minute on a laptop), robust and general purpose. Figure 1 illustrates the most significant steps.

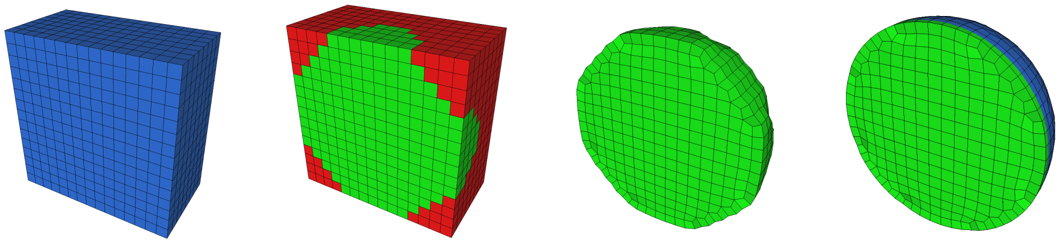

Fig. 1. Basic hexahedral meshing scheme: (from left to right) A grid that englobes the whole geometry is generated, the surface mesh is intersected with the grid and inside and outside subdomains are deduced. A buffer-layer is wrapped around the inner mesh and finally, the outer layer is projected on the real surface.

This method has been implemented in a commercial software named *Hexotic*, that is used by both academics and industrial customers for more than six years now. Consequently, it is fully tried and robust and is thus a good starting point for all-hexahedral BL generation. Useful information on octree or hexahedral meshing could be found in the following papers: [4], [5], [6], [1] and [3].

3. Mixing octree and boundary-layers

Such octree based meshes possess two interesting properties:

The first one is that a buffer-layer of elements wrapps around the surface so that any boundary hexahedron has only one face projected on the surface (see Figure 2). This single layer could be further subdivided to generate a set of several layers needed for calculation of physical phenomena near the boundary. This buffer-layer will further be referenced as the mother-layer as it will contain all the other boundary layers added for computational purposes.

The obvious advantage is that no growing process is needed as the main mother-layer already exists. One needs only to slice it into several sub-layers.

The obvious drawback is that the total thickness of layers could not exceed that of this mother-layer, which is a very limiting factor for some simulations like supersonic flows.

Furthermore, if one wishes to limit the layers to some portion of the surface, the way to properly end the layers has to be addressed. Here the second property comes into play: two surface quads from different surface subdomains are guaranteed to share only one edge. They cannot have only one vertex or two or more edges in common. Such property will prove very useful when generating the various ending patterns with hexahedra only (see Figure 3).

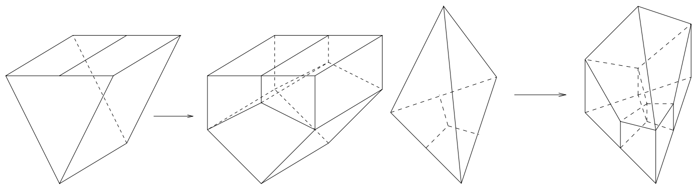

Fig. 2. Two cases of degenerate hexahedra near the boundary: before and after buffer layer insertion.

Adding BL generation capability to an octree-based hexahedral mesher is consequently pretty straightforward: it is a post-processing step that slices the mother-layer in several layers and does not require any modification to the main meshing algorithm.

However, generating good quality BL that follow all the user's requirement is quite complex and has to be broken down into several preparation steps that are presented in the following sections.

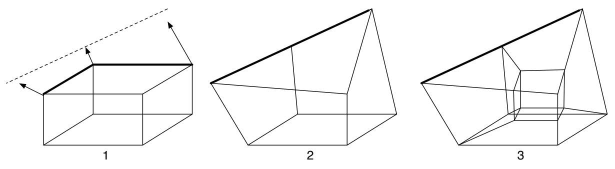

Fig. 3. (**1**) Initial cube with two edges to be projected on a ridge; (**2**) Degenerate element after projection; (**3**) Each hex of the buffer-layer has no more than one ridge-edge.

4. Checking and preparing the background mesh

Mother-layer inflating:. Since the total thickness of all layers cannot exceed that of the mother-layer it is important to make room as much as possible. To do that, the metric field associated with the hexahedra belonging to this layer is modified so that its aspect ratio is three to one in the direction orthogonal to the surface. Afterward, several smoothing steps are applied to the whole mesh so that the mother-layer's elements are inflated and the inner volume elements are pushed away from this layer, thus making room for the BL.

Size checking:. Since the user is free to specify as many layers of any thickness he wishes, the total layers size is arbitrary and could be thicker than the mother-layer in the area where the BL should be inserted. In this case, a reverse problem is solved so that the number of layers is maintained, but their size is reduced so that they all fit inside the mother-layer. On the other way, if the last layer's size is smaller that half the mother-layer's thickness, which would create a strong size gap between this last layer and the rest of the volume mesh. In this case, some additional layers are automatically added to mitigate the size transition. These are called blending layers.

Figure 4 illustrates why it is important to prepare the volume mesh before inserting the BL.

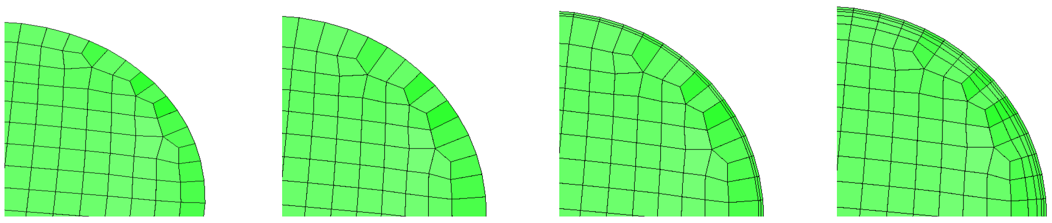

Fig. 4. Inserting boundary-layers in an existing hexahedral mesh (from left to right): initial hex mesh with an outer buffer-layer, the buffer-layer is inflated, then a first set of physical layers is inserted, then some blending layers are added to mitigate the size transition between the last layer and the rest of the volume mesh.

After this step has been performed, the volume mesh is ready for BL insertion.

5. Global generation scheme

The global generation scheme is itself quite complex and requires four different sub-steps.

At first, elements are tagged depending on the set of layers they belong to, or the kind of layer ending that should be used. Secondly, the vectors that will guide the generation of BL vertices are generated and their size and direction are adjusted to make sure that the final BL element volumes are positive. On a third step, only the BL vertices are generated along those previously defined vectors. And finally the BL elements are generated, connecting the BL vertices. The next four sections illustrate those steps.

6. Set surface quads properties

Both surface vertices and quads have to be properly qualified in order to generate the right kind of BL.

The analysis is first performed on the boundary quads, and after their properties have been set, surface vertex information is set accordingly.

Each boundary quad can belong to a surface subdomain that may generate BL or not. Each of those subdomains has a its own set of parameters that are the number and thickness of layers and how those layers should end: wedge, imprinting or cross-imprinting.

Topologically speaking, a quad is considered *inner* if it stands in the middle of a surface subdomain, which means that all of its four neighbors belong to the same subdomain. Conversely, it is qualified as *boundary* if one (and there may be only one as seen in Section 3) of its neighbors belongs to a different subdomain.

Inner quads fall into two simple sub-categories: they do not generate BL, or they generate a regular set of layers, that is, all of their four vertices are extruded and a set of hexahedra is orthogonally generated from the surface quad (see the first case in Figure 5).

Boundary quads are more complex and fall into four different sub-categories:

1. does not generate BL but a neighboring quad generates imprinting BL, consequently this quad will be imprinted by its neighbor's layers even though it does not itself generate layers itself (see the third case in Figure 5),
2. does generate regular BL but a neighboring quad does not generate BL, thus a wedge-shaped set of layers should be generated (see the second case in Figure 5),
3. does generate imprinting BL but a neighboring quad does not generate BL, thus a regular set of layers should be generated, but the pair of shared vertices will be extruded along the neighboring quad's surface and not orthogonally to the generating quad since the layers should imprint the surface (see third case in Figure 5),
4. does generate cross-imprinting BL and a neighboring quad does the same, both sets of layers will imprint each other creating a grid like mesh along the common edge (see rightmost case in Figure 5).

One has to keep in mind that inserting a layer of hexahedra is equivalent to inserting a sheet in the dual mesh as explained in [14]. We then have to make sure that every hexes have at most one surface quad and that BL quads have at most one edge shared with quads from different surface domains. If those conditions are not enforced, we will either end with arbitrary polyhedral elements, or the layer will not stop at the desired surface domain, but will propagate through the whole mesh.

Conversely, surface vertices belong to the two same main topological categories: inner vertices are shared only by quads that belong to the same subdomain and boundary vertices are those shared by quads from two different subdomains.

Inner vertices themselves fall into two simple sub-categories: those that do not generate BL and those that generate a regular set of layers.

Boundary vertices are also more complex and fall into three different sub-categories:

1. the vertex is shared by some quads that do not generate BL and some that generate wedge-shaped ending, consequently it does not generate BL as it stands on the sharp edge of the wedge (see second case in Figure 6),
2. the vertex is shared by some quads that do not generate BL, and some that generate imprinting layers, consequently this vertex is to be extruded along a path orthogonal to the boundary edge separating the two subdomains and that is coplanar with the imprinted quad (see third case in Figure 6),
3. the vertex is shared by quads from two subdomains that generate cross-imprinting layers, this is the most complicated case as the vertex is to be doubly extruded along a pair of vectors and will generate a 2D grid (see the rightmost case in Figure 6).

Since we know what to do for each BL quad and vertex, the next step is to set the vectors along which the BL vertices are to be extruded.

7. Generate and adjust the extrusion vectors

Before BL vertices can be generated, each vertex that has to be extruded should be associated with a vector along which the BL vertices will be projected. This vector should be carefully chosen so that the resulting BL elements are

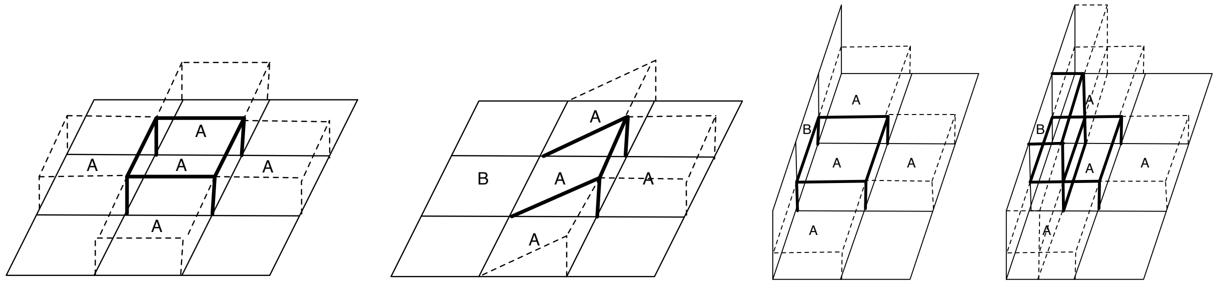

Fig. 5. Different kinds of BL quads (from left to right): regular, wedge, imprinting, cross-imprinting

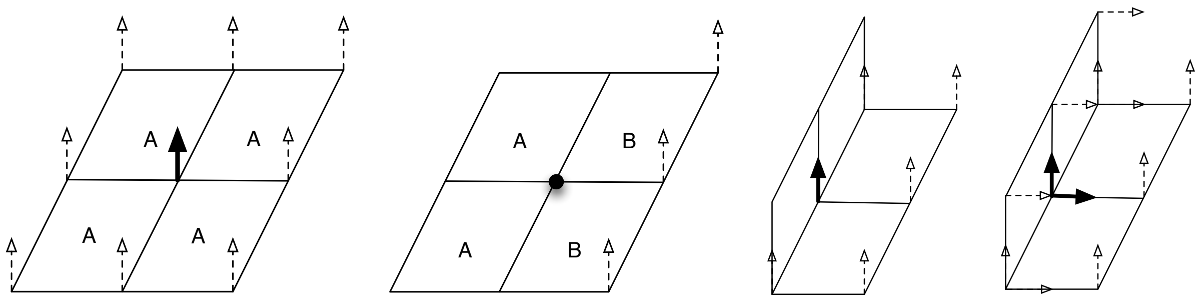

Fig. 6. Different kinds of BL vertices (from left to right): regular, wedge (no layers are generated), imprinting, cross-imprinting

valid. To do that, we first only set the vectors, simulate the shape of future layers and adjust the vectors to enhance the elements quality.

Setting the vectors:. For each surface vertex that will generate BL, the average ball-quads normal is computed (the ball of a vertex is the set of quads that share this common vertex). An additional operation is applied to imprinting or cross-imprinting vertices: the initial vector is projected on the imprinted plane.

Adjusting the vectors:. A loop is performed on BL quads: their four vertex vectors are retrieved so that it is possible to simulate the shape of the final layers (see Figure 8), the vector's size and direction are adjusted to avoid invalid BL hexes.

The BL elements quality is evaluated with the jacobian only and the two extra diagonal volumes used in *Hexotic* (see [3]) are not used. Although these two diagonal tetrahedra are useful to evaluate the hexahedron's twist, they degenerate very quickly when the element becomes very thin and the source boundary quad is not planar. Thus, one of the two volumes is very often negative and the only way to righten it up is to enforce planar quads on the surface, which is impossible for an arbitrary geometry cut from an octree structure. The resulting elements are valid in the jacobian sense, but they may be highly twisted.

After the extrusion vectors are properly set, we can now move forward to the BL vertex generation.

8. Generate the vertices

As seen in Section 6, there are two kinds of BL vertices, regular and cross-imprinting (see fig 9).

Growing the regular vertices is pretty straightforward, it consists in generating the physical and blending vertices along the guiding vector, respecting the required sizes and spacing factor defined in the previous step. This step is also very easy to implement as the BL vertices are stored in a linked list associated with the base vertex.

Things are getting more complex when it comes to cross-imprinting vertices as they generate a 2D analytical mesh along the two guiding vectors. This small grid has to be stored in a clever way since neighboring regular BL quads will need to access a subset of the grid in a particular order. One needs to develop a procedure to extract a regular set

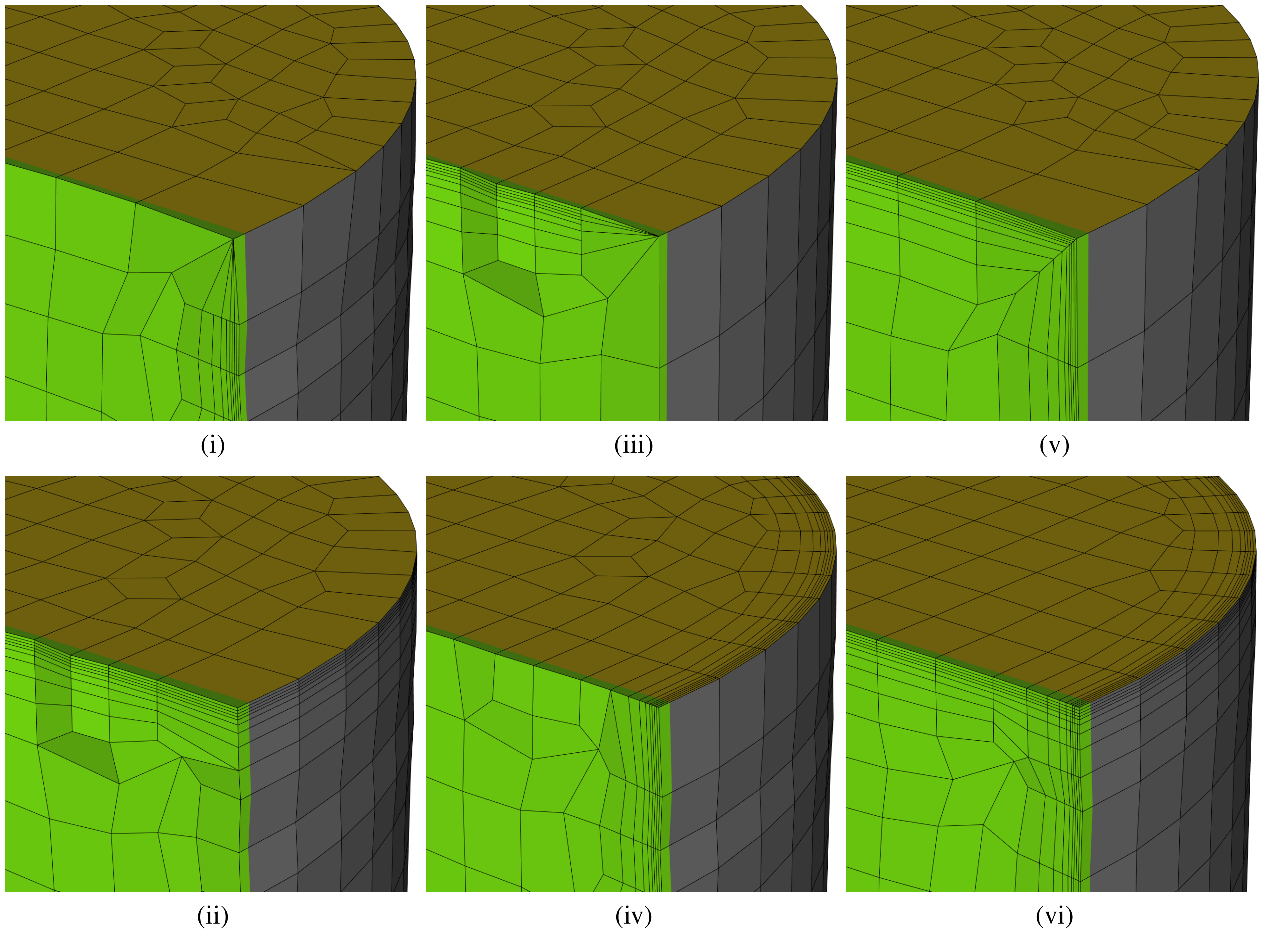

Fig. 7. Cylinder test case: Yellow surface generates BL that end in a wedge at grey surface **(i)**, or imprint the grey surface **(ii)**, grey surface generates BL that end in a wedge at yellow surface **(iii)**, or imprint the yellow surface **(vi)**, both surfaces generate BL and connect in a regular way **(v)**, or cross-imprint each others **(iv)**.

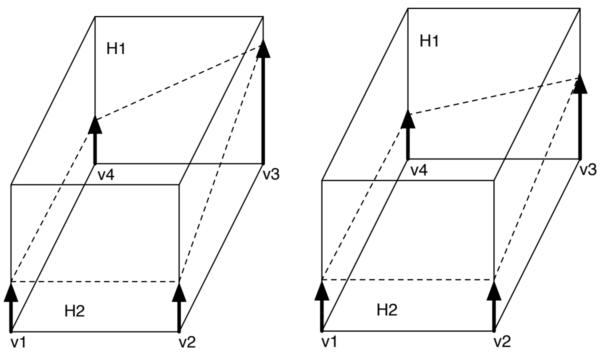

Fig. 8. Adjusting the vector size: **left:** vector v_3 is too long and the original hexahedron H_1 would become too distorted because of the short edge, **right:** the vector is gradually shortened until the two hexes reach a threshold quality.

of BL vertices from a cross-imprinting one, which is done by storing the row's and column reference surfaces along with the grid. When a single line of this grid is to be extracted as seen in figure 10 at left, the regular BL quad's ref is first compared to that of the grid column ref. If they are the same, the rightmost column is returned. If the quad's ref corresponds to the grid row reference, then the last row is returned.

The figure 10 on the right illustrates this process. Let's say that the blue face (F1) is processed before the red one (F3), triggering the creation of the cross-imprinting grid associated with vertex V1. Since the blue ref started the process, vertices along red lines are stored consecutively in the same column. When the face F2, which is a regular case BL, is processed, it requires BL vertices from its two vertices, V2 and V3. But V2 was previously generated by the face F1 and is stored in the last column of its grid. So face F2 starts a request process to grid V1 with blue as major reference and red as minor reference and will get the grid's right most column of vertices as a result. When comes the time to process face F3, nothing will be done since the other face F1 did the job. Finally, face F4 is processed and falls in the same regular case as F2, but this time the request to V1 grid is made with red as major reference and blue as minor. Consequently the topmost line of the grid is returned.

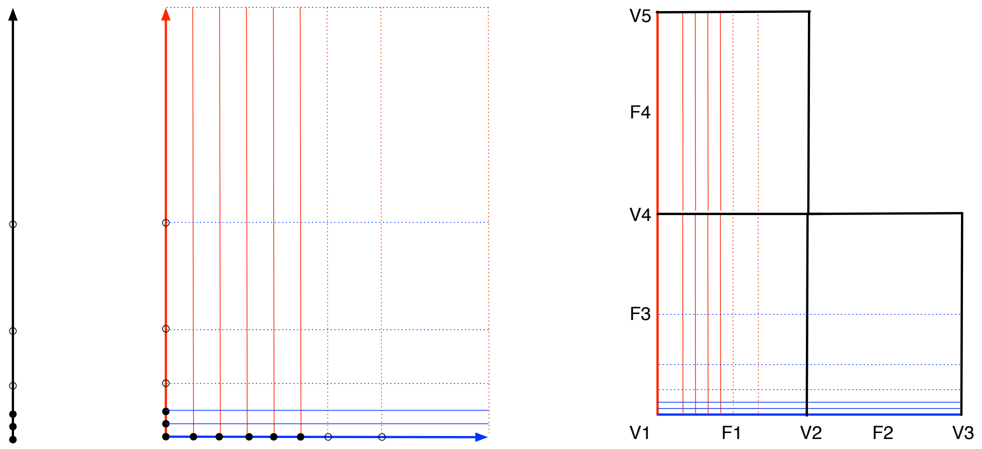

Fig. 9. Generating the BL vertices along vector paths: **left:** a regular vertex is extruded along a single path, dark dots are the first set of physical layers and white dots are the blending layers, **center:** a cross-imprinting vertex is extruded along two paths leading to a regular grid. Note that the numbers and sizes of layers may be different on each side, **right:** regular BL elements accessing a neighboring cross-imprinting grid.

Aside from the cross-imprinting grid generation, this step is very easy since all the trouble with elements quality has been dealt with in the previous step. We are now ready to move to the final BL elements generation step.

9. Generate the elements

As seen in Section 6, the BL quads fall into four different categories. However, they can be reduced to three since imprinting BL is just a regular BL with some constraints on the extrusion vectors. In a topological way both are identical.

That leaves us with three BL element generators to implement: regular, wedge and cross-imprinting.

Regular BL. A set of layers is extruded from the base quad along the vectors associated with each vertex (see the leftmost case in Figure 10). This the most common case.

Wedge shaped BL. Ending a set of BL with wedge elements could be done in two ways: either by using prismatic elements which would lead to a hybrid mesh, or by using degenerate hexahedra that look like prisms. The trick is to associate two consecutive hexahedral layers to one wedge shaped hexahedron as illustrated in the second case in Figure 10. In order to avoid those null jacobian elements because of the two collinear edges, the middle vertex could be slightly moved away for the sake of quality, but at the cost of loosing orthogonality to the boundary. Which of the three schemes is the best one depends on the solver that will compute on such meshes. It was suggested that this layer of wedges could be pushed one row of surface quads away from the BL generating surface so that the orthogonality would be maintained throughout the whole area of interest. This might be part of a next release.

Cross-imprinting BL. Such case is made up of two operations, one that generates a regular set of BL from the quad surface itself, and another one that generates a grid of cross-imprinting BL from the edge that separates the two cross-imprinting quads as seen in the rightmost case in Figure 10. The first operation is handled by the regular generator, but the second one needs a specific generator that is called only once by one of the two neighboring quads (usually the one with the greatest index). Note that when accessing the set of BL vertices from a cross imprinting grid, we need to extract the rightmost or topmost set column or lines of vertices as seen in Figure 9.

This second operation first retrieves the two grids from the two cross-imprinting edge vertices in the right order so that BL elements are properly oriented. After that, two nested loops are performed, one for each set of layers, so that the number of BL elements generated is the product of the number of layers from both sets.

Imprinting corners. This very particular case has not been presented so far. It occurs very rarely at vertices shared by three different subdomains: one that does not generate BL, one that generates regular BL and one that generates imprinting ones. While there may be only a few of them in a mesh, we need to address this situation anyway. Unfortunately, to do that we have to resort to pyramids as the transition between wedges and regular layers is too complex to be meshed with good quality hexahedra. If one wants to avoid hybrid elements at all cost, it is possible to propagate the imprinting surface to all neighboring surface quads as long as the dihedral angle they make is flat on convex until a new imprinting surface domain is defined so that it is only surrounded by concavities. In this way the imprinting layer will go all around this new domain and its neighbors, forming a closed loop, thus avoiding the creation of imprinting corners.

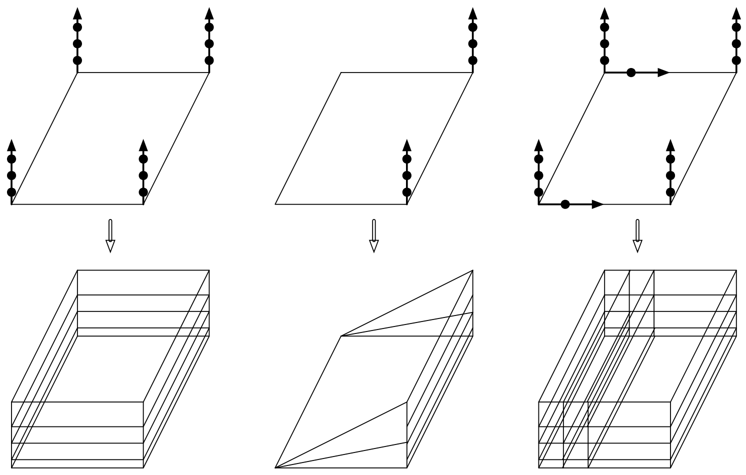

Fig. 10. Top row: three different kinds of BL quads with their extrusion vectors. Bottom row: corresponding BL meshes generated.

Unaddressed or poorly addressed matters. Multi-normals is indirectly addressed, since *Hexotic* cannot mesh angles sharper than 60° (see [3] for the complete meshing algorithm) and fillets are automatically added. One sharp edge is replaced by its two layers of adjacent surface quads on both sides. Those quads are consequently projected on a reconstructed surface that is a cylinder whose axis is the original sharp edge. Thus, four normal vectors stem out of the initial sharp node (see Figure 11).

Normal blending was not implemented since it is useful only when the BL grows very far away from the surface which is not possible with the current approach.

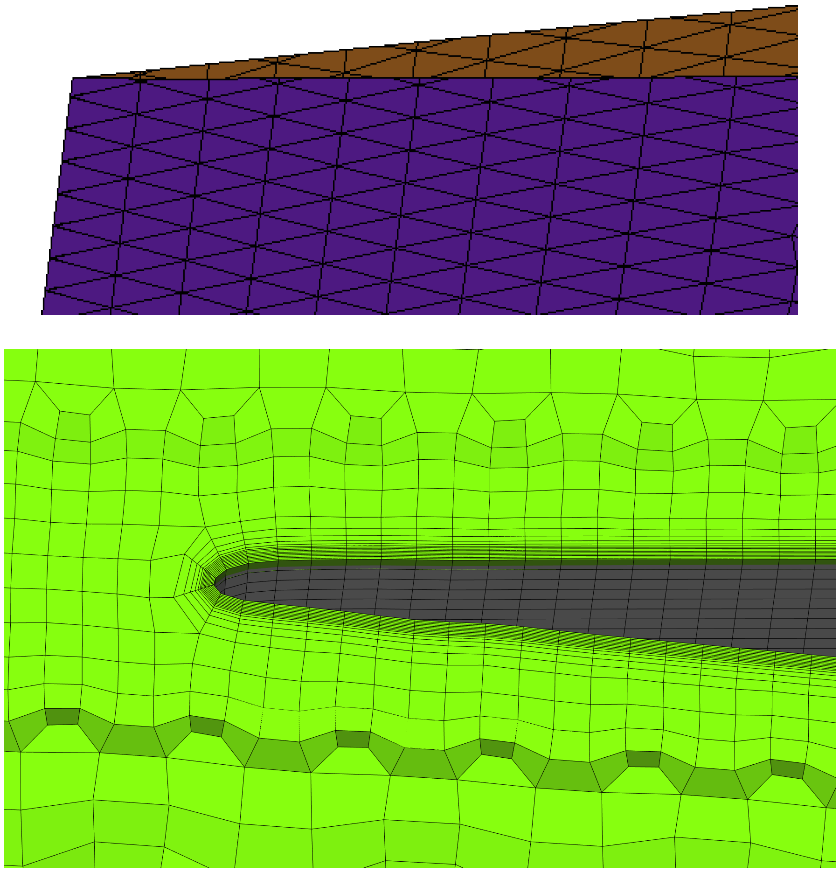

Fig. 11. **Top:** The input surface of a sharp wing trailing edge. **Bottom:** Cut through the resulting fillet hex mesh.

10. Conclusion

Adding a boundary-layers generation capacity to an octree hexahedral mesher proved feasible albeit with some limitation. The strongest being that the layers cannot grow very far away from the surface, and that this distance depends on the size of surface quads in the BL area. Aside from those strong limitations, all other useful features have been successfully implemented, even the most challenging ones like imprinting and cross-imprinting.

Two directions might be investigated to tackle those limitations. First, we may try to inflate the mother-layer much further by using a full-blown anisotropic smoothing scheme instead of the current implementation trick that imposes an anisotropic metric field only in the mother-layer but treats the rest of the volume mesh as isotropic. Doing that, we are able to stretch the first layer up to an aspect ration of 3 to 1 in the orthogonal direction, which is very limited. When we tried to push it further away, the rest of the volume mesh got squeezed and the smoothing algorithm, which is isotropic, tended to push the nodes back to their original position since it tried to make the element's shape as cubic as possible. It would take to generate and smooth a real anisotropic field in the whole geometry, which, after some trials, proved to be more challenging than we thought.

Another approach, similar to that in [2], and complementary to the first one, would be to inflate the mother layer as much as possible in every area, even if it leads to very different thicknesses depending of the local geometrical constrains and quad sizes. We could then modify the number of layers locally according to the local mother-layer thickness. Such "best effort" approach was very successful with tetrahedral meshing and might also be of interest with hexahedra, even if it requires much complex topological tools.

Overall, the process proved to be quite reliable and fully automated and is currently being tested by several industrial end users. We hope to come with some real life calculation on these meshes at the time of the conference.

11. Examples

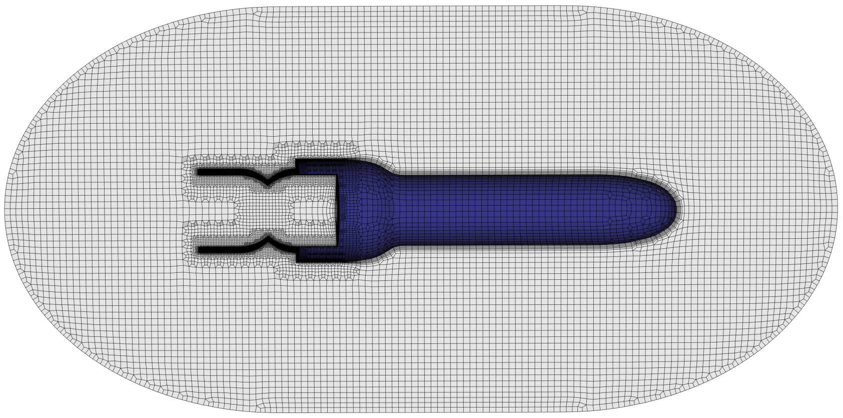
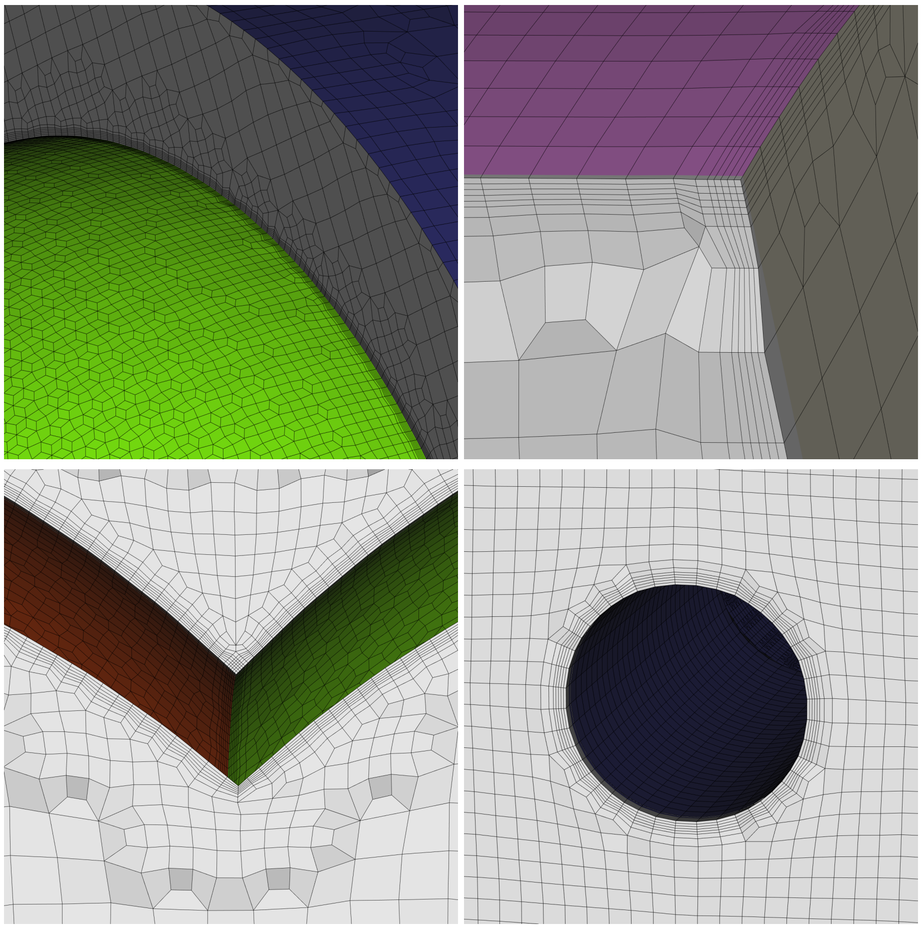

Fig. 12. Rocket test case (courtesy Dave Marcum, MSU), **top**: cut through the global view, **center left**: purple surface imprints the grey one, **center right**: green and grey surfaces cross-imprint each others, **bottom left**: zoom on the nozzle's inner part that features both a convex and a concave right angle that are meshed with cross-imprinting and regular BL respectively, **bottom right**: cross section through the body. The volume mesh is made up of 1.900.024 hexahedra (854.724 in the BL), scaled jacobian quality: average = 0.8579, minimum = 0.0451, meshing time = 12.4 seconds on a quad-core i7 laptop.

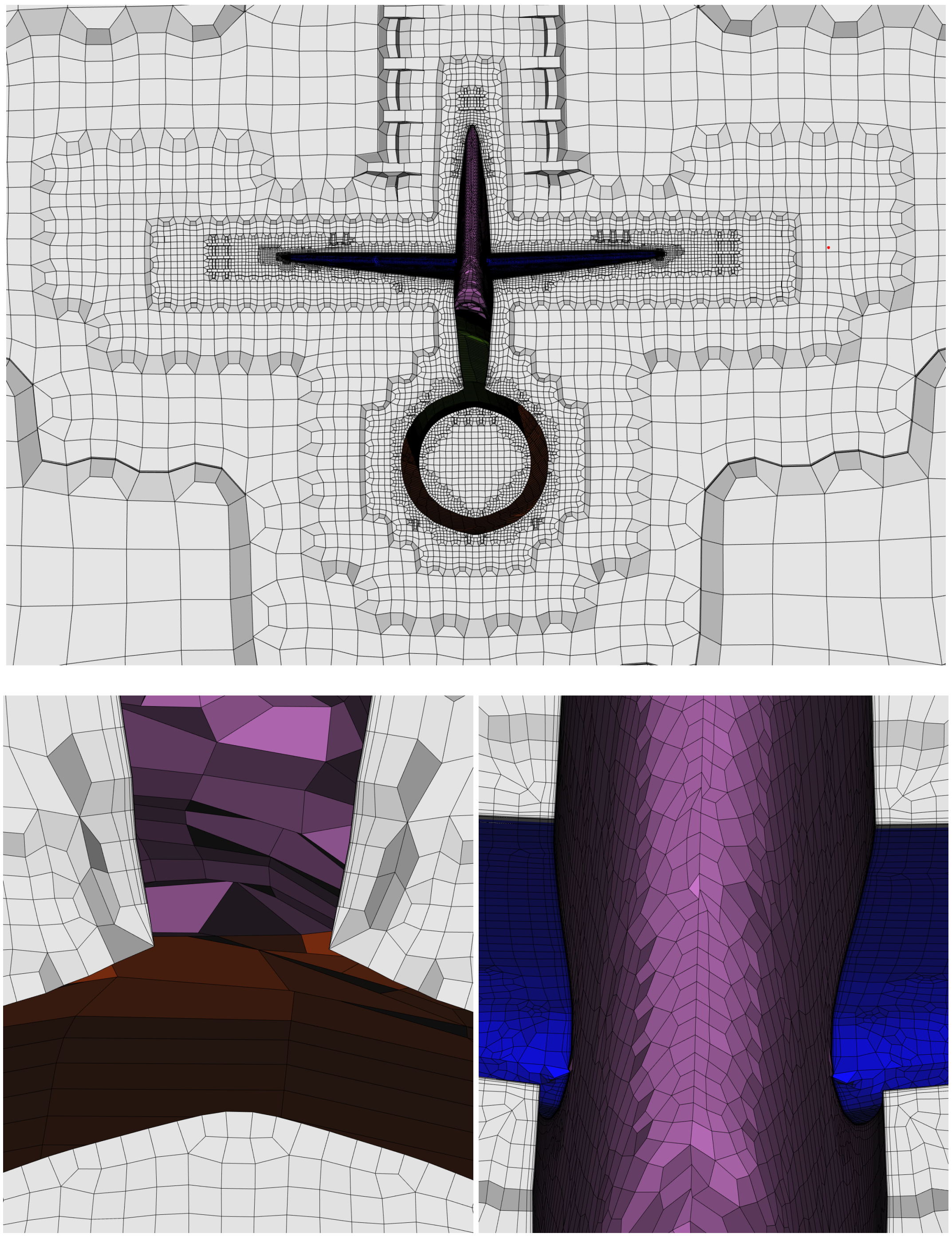

Fig. 13. Ouragan jet fighter (courtesy Dassault Aviation, 1954), **top**: cut through the tail's surface, **bottom left**: the tail's BL end on the body with a wedge, **bottom right**: tail and tail wing cross-imprint each others. The volume mesh is made up of 4.100.376 hexahedra (829,422 in the BL), scaled jacobian quality: average = 0.8490, minimum = 0.0001, meshing time = 37.4 seconds on a quad-core i7 laptop.

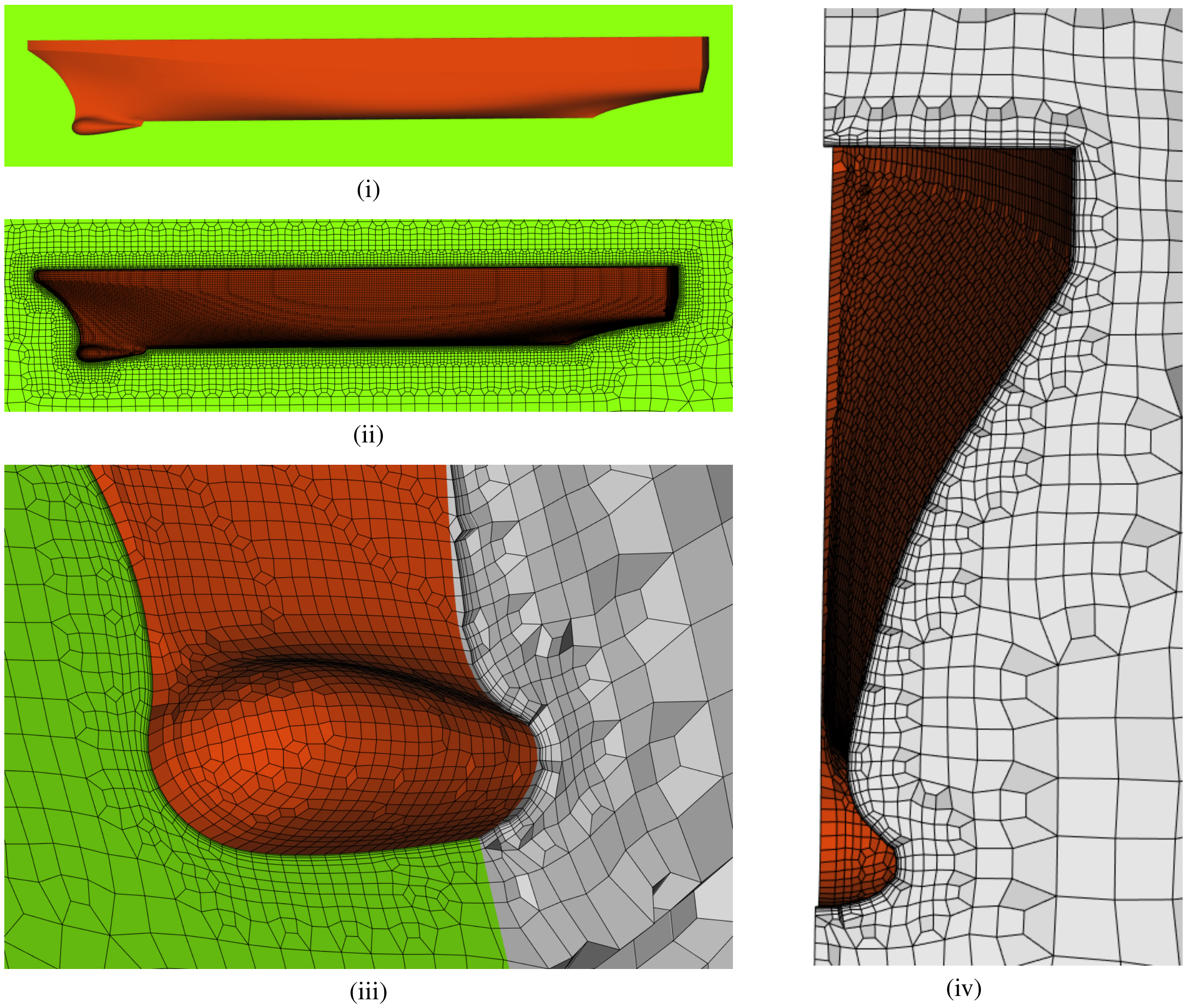

Fig. 14. DTMB ship hull, (test case provided by NextFlow Software for the Fortissimo project). The volume mesh is made up of 344,840 hexahedra (40,264 in the BL), scaled jacobian quality: average = 0.8427, minimum = 0.0312, the meshing time is 3.5 seconds on a quad-core i7 laptop. **(i)**: input surface mesh, **(ii)**: resulting quadrilateral surface mesh, **(iii)**: imprinting on the surface mesh around the keel, **(iv)**: cut through the BL and volume.

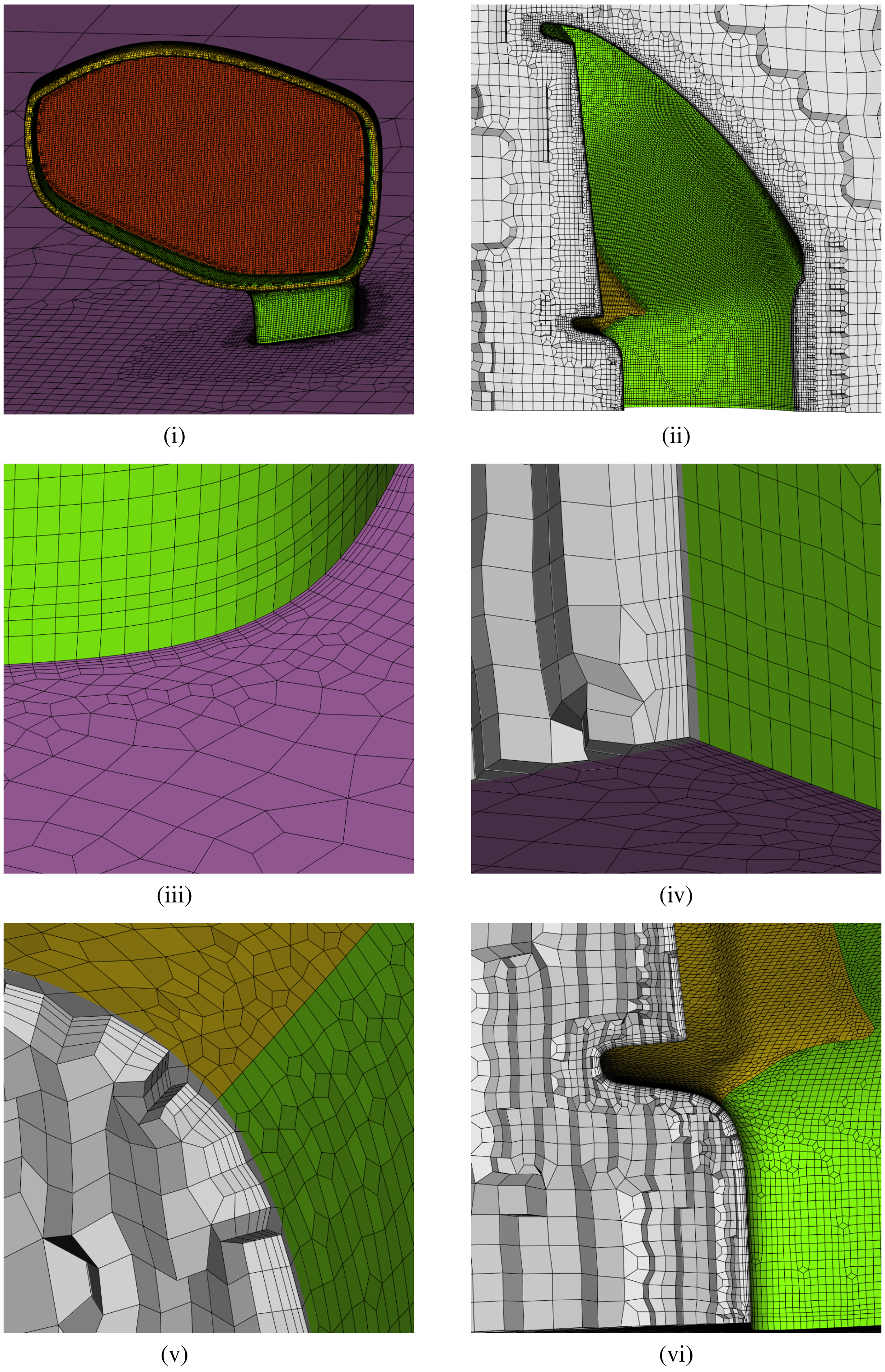

Fig. 15. Early development state of an AUDI A7 side mirror, (courtesy AUDI and I.A.G Universität Stuttgart). The volume mesh is made up of 1,012,583 hexahedra (393,708 in the BL), scaled jacobian quality: average = 0.8157, minimum = 0.0244, the meshing time is 8.5 seconds on a quad-core i7 laptop. **(i)**: surface mesh, **(ii)**: cut through the volume mesh, **(iii)**: imprinting on the surface mesh, **(iv)**: imprinting on the volume (grey) and surface (green imprints on violet), **(v)**: BL elements respect the surface subdomain boundary, **(vi)**: cut through the BL and volume.

References

[1] G. DHONT, *A new automatic hexahedral mesher based on cutting*, Int Jour Numer Meth Eng, Vol 50, pp 2109-2126, 2001.

[2] D.L. MARCUM & J.A. GAITHER, *Mixed element type unstructured grid generation for viscous flow applications*, 14th AIAA Computational Fluid Dynamics Conference.

[3] L. MARÉCHAL, *Advances in octree-based all-hexahedral mesh generation: handling sharp features*, Proc. IMR, *n*° 18, pp. 65-84, 2009.

[4] M.S. SHEPHARD & M.K. GEORGES, *Automatic three-dimensional mesh generation by the finite octree technique*, SCOREC Report *n*° 1, 1991.

[5] R. SCHNEIDERS & R. BÜNTEN, *Automatic generation of hexahedral finite element meshes*, Computer Aided Geometric Design, Vol. 12, pp. 693-707, 1995.

[6] R. SCHNEIDERS, *Octree-based hexahedral mesh generation*, Int. J. of Comp. Geom. & Applications, Vol. 10, *n*° 4, pp. 383-398, 2000.

[7] K. MERKLEY, C. ERNST, J. F. SHEPHERD, & M. J. BORDEN, *Methods and Applications of Generalized Sheet Insertion for Hexahedral Meshing*, Proceedings, 16th International Meshing Roundtable, pp.233-250, 2007.

[8] J. WACKERS, K. AIT SAID, G.B. DENG, P. QUEUTEY, M. VISONNEAU & I. MIZINE, *Adaptive Grid Refinement Applied to RANS Ship Flow Computation*, 28th Symposium on Naval Hydrodynamics Pasadena, California, 12-17 September 2010

[9] A. LOSEILLE & R. LÖHNER, *Boundary layer mesh generation and adaptivity*, 49th AIAA Aerospace Sciences Meeting, 2011.

[10] R. JAIN & T. J. TAUTGES, *PostBL: Post-mesh Boundary Layer Mesh Generation Tool*, 22nd International Meshing Roundtable, pp.331-348, 2013.

[11] Y. ITO, A.M. SHIH & B.K. SONI, *Efficient hybrid surface/volume mesh generation using suppressed marching-direction method*, AIAA Journal, Vol. 51, *n*° 6, pp. 1450-1461, 2013.

[12] F. ALAUZET & D. MARCUM, *A closed advancing-layer method with connectivity optimization based mesh movement for viscous mesh generation*, Eng. w. Comp., Vol. 31, pp. 545-560, 2015.

[13] M. Muller-Hannemann, *Hexahedral Mesh Generation by Successive Dual Cycle Elimination*, Proceedings, 7th International Meshing Roundtable, pp.365-378, 1998.

[14] N. J. Harris, S. E. Benzley, S. J. Owen, *Conformal refinement of all-hexahedral element meshes based on multiple twist plane insertion*, Proceedings, 13th International Meshing Roundtable, 2004.